CS 106A, Lecture 26
Final Exam Review 2
Plan for today

• Announcements
• HashMaps
• Classes
• Interactors
• Final Exam Tips
Plan for today

• Announcements
• HashMaps
• Classes
• Interactors
• Final Exam Tips
Announcements

• Final Exam OAE Accommodations
• Midterm Regrade Requests
• End-Quarter LaIR and Office Hours
• Graphics Contest (*drum roll *)
Winner: Aesthetics

Elina Thadhani
Winner: Algorithmics

Julianne Crawford
Winner: People’s Choice

Nicolas Guillen Barrail
Plan for today

- Announcements
- HashMaps
- Classes
- Interactors
- Final Exam Tips
# Review: HashMaps

<table>
<thead>
<tr>
<th></th>
<th>String</th>
<th>Array</th>
<th>2D Array</th>
<th>ArrayList</th>
<th>HashMap</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td>Sequence of letters or symbols</td>
<td>Fixed length elements in a list</td>
<td>Grid / Matrix of elements</td>
<td>Growable list of elements</td>
<td>Key/Value mapping</td>
</tr>
<tr>
<td><strong>Type of element</strong></td>
<td>chars</td>
<td>Objects &amp; Primitives</td>
<td>Objects &amp; Primitives</td>
<td>Objects</td>
<td>Object/Object</td>
</tr>
<tr>
<td><strong>Access Elements</strong></td>
<td>str.charAt(i);</td>
<td>arr[i];</td>
<td>arr[r][c];</td>
<td>list.get(i); list.set(i, elem)</td>
<td>map.put(key, value) map.get(key);</td>
</tr>
<tr>
<td><strong>Special notes</strong></td>
<td>Immutable</td>
<td>Watch bounds!</td>
<td>Row, col structure</td>
<td>Just fantastic</td>
<td>Each key must be unique.</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>”Hello world”</td>
<td>Histogram</td>
<td>ImageShop pixels</td>
<td>Hangman words, entries in namesurfer</td>
<td>NSDatabase, FPDatabase</td>
</tr>
</tbody>
</table>

- **Immutable**
- **Watch bounds!**
- **Row, col structure**
- **Just fantastic**
- **Each key must be unique. Unordered**
# Review: HashMaps

<table>
<thead>
<tr>
<th>Model</th>
<th>String</th>
<th>Array</th>
<th>2D Array</th>
<th>ArrayList</th>
<th>HashMap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequence of letters or symbols</td>
<td>Fixed length elements in a list</td>
<td>Grid / Matrix of elements</td>
<td>Growable list of elements</td>
<td>Key/Value mapping</td>
</tr>
<tr>
<td>Type of element</td>
<td>chars</td>
<td>Objects &amp; Primitives</td>
<td>Objects &amp; Primitives</td>
<td>Objects</td>
<td>Object/Object</td>
</tr>
<tr>
<td>Access Elements</td>
<td>str.charAt(i);</td>
<td>arr[i];</td>
<td>arr[r][c];</td>
<td>list.get(i);</td>
<td>map.put(key, value) map.get(key);</td>
</tr>
<tr>
<td>Special notes</td>
<td>Immutable</td>
<td>Watch bounds!</td>
<td>Row, col structure</td>
<td>Just fantastic</td>
<td>Each key must be unique. Unordered</td>
</tr>
<tr>
<td>Examples</td>
<td>&quot;Hello world&quot;</td>
<td>Histogram</td>
<td>ImageShop pixels</td>
<td>Hangman words, entries in namesurfer</td>
<td>NSDatabase</td>
</tr>
</tbody>
</table>
Review: HashMaps

• A variable type that represents a collection of key-value pairs
• You access values by *key*, and all keys are *unique*
• Keys and values can be any type of *object* (use wrapper classes to store primitives)
• Resizable – can add and remove pairs
• Has a variety of methods you can use, including `.containsKey`, `.put`, `.get`, etc.
Key Idea: Association

- **Phone book**: name \(\rightarrow\) phone number
- **Search engine**: URL \(\rightarrow\) webpage
- **Dictionary**: word \(\rightarrow\) definition
- **Bank**: account # \(\rightarrow\) balance
- **Social Network**: name \(\rightarrow\) profile
- **Counter**: text \(\rightarrow\) # occurrences
- And many more...
Review: HashMap Operations

- `m.put(key, value);` Adds a key/value pair to the map.
  
  ```java
  m.put("Eric", "650-123-4567");
  ```

  - Replaces any previous value for that key.

- `m.get(key)` Returns the value paired with the given key.
  
  ```java
  String phoneNum = m.get("Jenny"); // "867-5309"
  ```

  - Returns null if the key is not found.

- `m.remove(key);` Removes the given key and its paired value.
  
  ```java
  m.remove("Rishi");
  ```

  - Has no effect if the key is not in the map.

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Jenny&quot;</td>
<td>&quot;867-5309&quot;</td>
</tr>
<tr>
<td>&quot;Mehran&quot;</td>
<td>&quot;123-4567&quot;</td>
</tr>
<tr>
<td>&quot;Marty&quot;</td>
<td>&quot;685-2181&quot;</td>
</tr>
<tr>
<td>&quot;Chris&quot;</td>
<td>&quot;947-2176&quot;</td>
</tr>
</tbody>
</table>
Review: HashMap Operations

- `m.containsKey(key);` Returns true if the key is in the map, false otherwise

- `m.size();` Returns the number of key/value pairs in the map.

To iterate over a map:

```java
for (KeyType key : map.keySet()) {
    ValueType value = map.get(key);
    // Do something with key and/or value
}
```
What data structure should I use?

• Use an **array** if...
  – Order matters for your information
  – You know how many elements you will store
  – You need the most efficiency

• Use an **ArrayList** if...
  – Order matters for your information
  – You do not know how many elements you will store, or need to resize
  – You need to use ArrayList methods

• Use a **HashMap** if...
  – Order doesn’t matter for your information
  – You need to store an *association* between two types of information
  – You do not know how many elements you will store, or need to resize
  – You need to use HashMap methods
Practice: Anagrams

• Write a program to find all **anagrams** of a word the user types.

  Type a word [Enter to quit]: **scared**
  Anagrams of scared:
  cadres cedars sacred scared

• Assume you are given a **dictionary.txt** file containing words in the dictionary.

• How can a HashMap help us solve this problem?
Key Idea: Anagrams

• Every word has a *sorted form* where its letters are arranged into alphabetical order.
  
  "fare" → "aefr"
  "fear" → "aefr"
  "swell" → "ellsw"
  "wells" → "ellsw"

• Notice that anagrams have the same *sorted form* as each other.

• Assume we provide a **String sortLetters(String s)** method that takes a string and returns the string with its characters alphabetically ordered.
public void run() {
    HashMap<String, ArrayList<String>> anagrams = createAnagramsMap();

    // prompt user for words and look up anagrams in map
    while (true) {
        String word = readLine("Type a word [Enter to quit]: ");
        if (word.length() == 0) {
            break;
        }

        String sorted = sortLetters(word.toLowerCase());
        if (anagrams.containsKey(sorted)) {
            println("Anagrams of " + word + ":");
            println(anagrams.get(sorted));
        } else {
            println("No anagrams for " + word + ".");
        }
    }
}
public void run() {
    HashMap<String, ArrayList<String>> anagrams = createAnagramsMap();

    // prompt user for words and look up anagrams in map
    while (true) {
        String word = readLine("Type a word [Enter to quit]: ");
        if (word.length() == 0) {
            break;
        }

        String sorted = sortLetters(word.toLowerCase());
        if (anagrams.containsKey(sorted)) {
            println("Anagrams of " + word + ":");
            println(anagrams.get(sorted));
        } else {
            println("No anagrams for " + word + ".");
        }
    }
}
public void run() {
    HashMap<String, ArrayList<String>> anagrams = createAnagramsMap();

    // prompt user for words and look up anagrams in map
    while (true) {
        String word = readLine("Type a word [Enter to quit]: ");
        if (word.length() == 0) {
            break;
        }

        String sorted = sortLetters(word.toLowerCase());
        if (anagrams.containsKey(sorted)) {
            println("Anagrams of " + word + ":");
            println(anagrams.get(sorted));
        } else {
            println("No anagrams for " + word + ".");
        }
    }
}
Anagrams Solution

```java
public void run() {
    HashMap<String, ArrayList<String>> anagrams = createAnagramsMap();

    // prompt user for words and look up anagrams in map
    while (true) {
        String word = readLine("Type a word [Enter to quit]: ");
        if (word.length() == 0) {
            break;
        }

        String sorted = sortLetters(word.toLowerCase());
        if (anagrams.containsKey(sorted)) {
            println("Anagrams of " + word + ":");
            println(anagrams.get(sorted));
        } else {
            println("No anagrams for " + word + ".");
        }
    }
}
```
public void run() {
    HashMap<String, ArrayList<String>> anagrams = createAnagramsMap();

    // prompt user for words and look up anagrams in map
    while (true) {
        String word = readLine("Type a word [Enter to quit]: ");
        if (word.length() == 0) {
            break;
        }

        String sorted = sortLetters(word.toLowerCase());
        if (anagrams.containsKey(sorted)) {
            println("Anagrams of " + word + ":");
            println(anagrams.get(sorted));
        } else {
            println("No anagrams for " + word + ".");
        }
    }
}
public void run() {
    HashMap<String, ArrayList<String>> anagrams = createAnagramsMap();

    // prompt user for words and look up anagrams in map
    while (true) {
        String word = readLine("Type a word [Enter to quit]: ");
        if (word.length() == 0) {
            break;
        }

        String sorted = sortLetters(word.toLowerCase());
        if (anagrams.containsKey(sorted)) {
            println("Anagrams of " + word + ":");
            println(anagrams.get(sorted));
        } else {
            println("No anagrams for " + word + ".");
        }
    }
}
/*
* This method reads in the dictionary.txt file and creates a map
* from a sorted string of characters to all words made up of those
* characters. e.g. “acers” - {“scare”, “cares”,...}
*/

private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams = new HashMap<String, ArrayList<String>>();

    try {
        Scanner scanner = new Scanner(new File("dictionary.txt"));
        while (scanner.hasNextLine()) {
            String word = scanner.nextLine();
            String sorted = sortLetters(word);
            ...
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }

    return anagrams;
}
/*
 * This method reads in the dictionary.txt file and creates a map
 * from a sorted string of characters to all words made up of those
 * characters. e.g. “acers” -> {“scare”, “cares”,...}
 */

private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams = new HashMap<>();

    try {
        Scanner scanner = new Scanner(new File("dictionary.txt"));
        while (scanner.hasNextLine()) {
            String word = scanner.nextLine();
            String sorted = sortLetters(word);
            ...
        }
    }
}
/*
 * This method reads in the dictionary.txt file and creates a map
 * from a sorted string of characters to all words made up of those
 * characters. e.g. “acers” -> {“scare”, “cares”,...}
 */
private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams = new HashMap<String, ArrayList<String>>();

    try {
        Scanner scanner = new Scanner(new File("dictionary.txt"));
        while (scanner.hasNextLine()) {
            String word = scanner.nextLine();
            String sorted = sortLetters(word);
            ...
        }
    }
}
private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams = new HashMap<String, ArrayList<String>>();
    try {
        Scanner scanner = new Scanner(new File("dictionary.txt"));
        while (scanner.hasNextLine()) {
            String word = scanner.nextLine();
            String sorted = sortLetters(word);
            ...
        }
    } catch (FileNotFoundException e) {
        System.err.println("File not found: dictionary.txt");
    }
    return anagrams;
}
Anagrams Solution, Part 2

/*
 * This method reads in the dictionary.txt file and creates a map
 * from a sorted string of characters to all words made up of those
 * characters. e.g. “acers” -> {“scare”, “cares”,...}
 */
private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams = new HashMap<String, ArrayList<String>>();

    try {
        Scanner scanner = new Scanner(new File("dictionary.txt"));
        while (scanner.hasNextLine()) {
            String word = scanner.nextLine();
            String sorted = sortLetters(word);
            ...
        }
    }
}
/** Either get the current word list, * or make a new empty one. */
ArrayList<String> words;
if (anagrams.containsKey(sorted)) {
    words = anagrams.get(sorted);
} else {
    words = new ArrayList<String>();
}

// Update the list with our new word
words.add(word);
anagrams.put(sorted, words);

scanner.close();
} catch (IOException ex) {
    println("Error reading file dictionary.txt.");
}

return anagrams;
Either get the current word list, *or make a new empty one.*

```java
ArrayList<String> words;
if (anagrams.containsKey(sorted)) {
    words = anagrams.get(sorted);
} else {
    words = new ArrayList<String>();
}
```

// Update the list with our new word
words.add(word);
anagrams.put(sorted, words);
```
/* Either get the current word list, * or make a new empty one. */
ArrayList<String> words;
if (anagrams.containsKey(sorted)) {
    words = anagrams.get(sorted);
} else {
    words = new ArrayList<String>();
}

// Update the list with our new word
words.add(word);
anagrams.put(sorted, words);

scanner.close();
} catch (IOException ex) {
    println("Error reading file dictionary.txt.");
}

return anagrams;
Anagrams Solution, Part 2

/* Either get the current word list, *
 * or make a new empty one. */
ArrayList<String> words;
if (anagrams.containsKey(sorted)) {
    words = anagrams.get(sorted);
} else {
    words = new ArrayList<String>();
}

// Update the list with our new word
words.add(word);
anagrams.put(sorted, words);
}

scanner.close();
}
} catch (IOException ex) {
    println("Error reading file dictionary.txt.");
}

return anagrams;
Anagrams Solution, Part 2

/* Either get the current word list, 
* or make a new empty one. */
ArrayList<String> words;
if (anagrams.containsKey(sorted)) {
    words = anagrams.get(sorted);
} else {
    words = new ArrayList<String>();
}

// Update the list with our new word
words.add(word);
anagrams.put(sorted, words);
}
scanner.close();
} catch (IOException ex) {
    println("Error reading file dictionary.txt.");
}

return anagrams;
Plan for today

• Announcements
• HashMaps
• Classes
• Interactors
• Final Exam Tips
A class defines a new variable type.
Classes Are Like Blueprints

iPod blueprint (class)

**state:**
- current song
- volume
- battery life

**behavior:**
- power on/off
- change station/song
- change volume
- choose random song

---

iPod (variable) #1

**state:**
- song = "1,000,000 Miles"
- volume = 17
- battery life = 2.5 hrs

**behavior:**
- power on/off
- change station/song
- change volume
- choose random song

iPod (variable) #2

**state:**
- song = "Letting You"
- volume = 9
- battery life = 3.41 hrs

**behavior:**
- power on/off
- change station/song
- change volume
- choose random song

iPod (variable) #3

**state:**
- song = "Discipline"
- volume = 24
- battery life = 1.8 hrs

**behavior:**
- power on/off
- change station/song
- change volume
- choose random song
Creating A New Class

1. What information is inside this new variable type? These are its instance variables.

2. What can this new variable type do? These are its public methods.

3. How do you create a variable of this type? This is the constructor.
Methods defined in classes can be called on an instance of that class.

When one of these methods executes, it can reference that object’s copy of instance variables.

```
ba1.deposit(0.20);
ba2.deposit(1000.00);
```

This means calling one of these methods on different objects has different effects.
The constructor is executed when a new object is created.
public class BankAccount {
    // Step 1: the data inside a BankAccount
    private String name;
    private double balance;

    // Step 2: the things a BankAccount can do (omitted)
    // Step 3: how to create a BankAccount
    public BankAccount(String accountName, double startBalance) {
        name = accountName;
        balance = startBalance;
    }

    public BankAccount(String accountName) {
        name = accountName;
        balance = 0;
    }
}
Using Constructors

BankAccount ba1 =
new BankAccount("Marty", 1.25);

BankAccount ba2 =
new BankAccount("Mehran", 900000.00);

• When you call a constructor (with `new`):
  – Java creates a new object of that class.
  – The constructor runs, on that new object.
  – The newly created object is returned to your program.
Practice: Airplane!

Let’s write a class called **Airplane** that implements functionality for boarding/unboarding passengers from a plane.

```java
int capacity = readInt("Capacity? ");
Airplane plane = new Airplane(capacity);

// Board passengers
while (!plane.isFull()) {
    String passengerName = readLine("Name: ");
    boolean priority = readBoolean("Priority? (true/false) ");
    plane.boardPassenger(passengerName, priority);
}

// fly...

// Unboard passengers
while (!plane.isEmpty()) {
    String passengerName = plane.unboardPassenger();
    println("Unboarded "+ passengerName);
}
```
Let’s write a class called **Airplane** that implements the following functionality for boarding/unboarding passengers from a plane.

```java
// Creates a new airplane with the given capacity
public Airplane(int capacity);

/* Boards 1 passenger, at front if they are priority, or
 * back otherwise */
public void boardPassenger(String name, boolean priority);

public boolean isFull();
public boolean isEmpty();

/* Unboards and returns next passenger, or null if there
 * are no more passengers. */
public String unBoardPassenger();
```
1. What information is inside this new variable type? These are its instance variables.

2. What can this new variable type do? These are its public methods.

3. How do you create a variable of this type? This is the constructor.
public class Airplane {
  private ArrayList<String> passengers;
  private int capacity;

  ...

Creating A New Class

1. What information is inside this new variable type? These are its instance variables.

2. What can this new variable type do? These are its public methods.

3. How do you create a variable of this type? This is the constructor.
... public void boardPassenger(String name, boolean priority) {
    if (!isFull()) {
        if (priority) {
            passengers.add(0, name);
        } else {
            passengers.add(name);
        }
    }
}
...
public boolean isFull() {
    return capacity == passengers.size();
}

...
public boolean isEmpty() {
    return passengers.isEmpty();
}
...
public String unboardPassenger() {
    if (!isEmpty()) {
        return passengers.remove(0);
    }
    return null;
}
Creating A New Class

1. What information is inside this new variable type? These are its instance variables.

2. What can this new variable type do? These are its public methods.

3. How do you create a variable of this type? This is the constructor.
// Private instance variables
private ArrayList<String> passengers;
private int capacity;

// Constructor
public Airplane(int numSeats) {
    capacity = numSeats;
    passengers = new ArrayList<String>();
}
...

Plan for today

• Announcements
• HashMaps
• Classes
• Interactors
• Final Exam Tips
Review: Interactors

1. Add interactors in `init()`
2. `addActionListeners()` to listen for button presses
3. `.addActionListener(this)` on text fields for ENTER
   1. Plus (usually) `setActionCommand(command)`
4. Implement `actionPerformed`
5. Java will call `actionPerformed` whenever an action event occurs.
public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equals("My Interactor")) {

    }
}

// ... equivalent to ...
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == myInteractor) {

    }
}
Practice: SplatterPaint
Practice: SplatterPaint

• Add a new “splatter” (e.g. GOval) every time the user clicks on the “Yellow” or “Orange” buttons.
• If the user hits “ENTER” instead, add a Yellow splatter.
• The splatter’s diameter should be what is entered in the text box (assume it’s valid), and placed randomly entirely onscreen.
• If the user clicks “Randomize”, all onscreen splatters should be changed to different random colors.
Solution: SplatterPaint

- If “Yellow” is clicked OR IF ENTER IS PRESSSED, add a splatter with the diameter in the text field to the screen in **yellow**
- If “Orange” is clicked, add a splatter with the diameter in the text field to the screen in **orange**
- If “Randomize” is clicked, iterate over all splatters and change their color
public class SplatterPaint extends GraphicsProgram {
    // The width of the diameter text field (in chars)
    private static final int DIAMETER_FIELD_WIDTH = 15;

    // The field for entering the splatter diameter
    private JTextField diameterField;

    // A list of all onscreen splatters
    private ArrayList<GOval> splatters;

    ...
}

public void init() {
    splatters = new ArrayList<GOval>();

    // Add the text field
    add(new JLabel("Diameter: "), SOUTH);
    diameterField = new JTextField(DIAMETER_FIELD_WIDTH);
    diameterField.setActionCommand("Yellow");
    diameterField.addActionListener(this);
    add(diameterField, SOUTH);

    // Add the buttons
    add(new JButton("Yellow"), SOUTH);
    add(new JButton("Orange"), SOUTH);
    add(new JButton("Randomize"), SOUTH);
    addActionListeners();
}
public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equals("Yellow")) {
        // Add a yellow circle
        int diameter =
            Integer.parseInt(diameterField.getText());
        addSplatter(diameter, Color.yellow);
    } else if (e.getActionCommand().equals("Orange")) {
        // Add an orange circle
        int diameter =
            Integer.parseInt(diameterField.getText());
        addSplatter(diameter, Color.orange);
    } else if (e.getActionCommand().equals("Randomize")) {
        // Randomize all existing splatters
        randomizeColors();
    }
}
private void addSplatter(int diameter, Color color) {
    int x = RandomGenerator.getInstance().nextInt(0, getWidth() - diameter);
    int y = RandomGenerator.getInstance().nextInt(0, getHeight() - diameter);

    GOval splatter = new GOval(x, y, diameter, diameter);
    splatter.setFilled(true);
    splatter.setColor(color);
    add(splatter);
    splatters.add(splatter);
}
private void randomColors() {
    for (G0val splatter : splatters) {
        Color newColor =
            RandomGenerator.getInstance().nextColor();
        splatter.setColor(newColor);
    }
}
Recap

• Announcements
• HashMaps
• Classes
• Interactors
• Final Exam Tips
Final Exam Tips

• Look over all problems before starting
• Pseudocode!
• Show your work
• Practice practice practice
• Review common programming paradigms
  – Iterating over a string
  – actionPerformed
  – ...

Final Exam Tips

• Reflect on how much you’ve learned in just 1 quarter!